An Engineering Information Service Infrastructure for Ubiquitous Computing

David Liu[1], Jinxing Cheng[2], Kincho H. Law[3], Gio Wiederhold[4], and Ram D. Sriram[5]

Abstract

This paper describes a software framework for the development of a ubiquitous computing environment for distributed engineering information services. Two fundamental issues are addressed: universal accessibility from devices to information services, and collaboration among the parties accessing the information services. The first issue calls for the development of device independent information services that have the flexibility to support wide range of client devices. We introduce a mediation-based framework that enables the information clients to calibrate the source information services to the clients' characteristics. The second issue requires effective integration of information services, for which we address in two ways: (1) we sketch an ontology standard and describe how such standard can be effectively applied for exchanging scheduling information; (2) we illustrate an infrastructure that is particularly suitable for the integration of engineering services. A prototype for the ubiquitous computing environment has been developed that incorporates a variety of project management software as well as different devices ranging from PDA, web browsers, desktop computers, and servers.

[1] Ph.D. Candidate, Department of Electrical Engineering, Stanford University, Stanford, CA 94305. E-mail: davidliu@stanford.edu
[2] Ph.D. Candidate, Department of Civil and Environmental Engineering, Stanford University, Stanford, CA 94305. E-mail: cjx@stanford.edu
[3] Professor, Department of Civil and Environmental Engineering, Stanford University, Stanford, CA 94305. E-mail: law@stanford.edu
[4] Professor, Computer Science Department, Stanford University, Stanford, CA 94305. E-mail: gio@db.stanford.edu
[5] Group Leader, Manufacturing Systems Integration Division, National Institute of Standards and Technology, Gaithersburg, MD 20899. E-mail: sriram@cme.nist.gov

1 Introduction

A shift to distributed computing is underway. Rapid proliferation of Internet protocols, fast expanding computing power, coupled with broadband and mobile communication technologies make truly ubiquitous computing possible for the first time. We will soon have an interconnected web of small devices that provide information to people regardless of their locations. However, ubiquitous computing is more than simply tying many wired and wireless gadgets together. Everything from client devices, to communication networks, to software applications needs to work together to enable two main characteristics of ubiquitous computing: (1) universal accessibility from any device to any information service, and (2) effective collaboration among the parties accessing information services.

Ubiquitous computing can find many applications in the engineering industry. The design and construction industries, such applications range from field inspection, to site procurement of materials, to interactive on-site project planning. For instance, with mobile devices, one could readily compare the as-built site condition with the planned design information, enquire availability of materials and receive immediate response to change orders, and gain dynamic interactions with Internet-based services.

A Ubiquitous computing environment is highly desirable in the A/E/C (Architecture/Engineering/Construction) industry. Productivity is hampered due to the lack of effective channels for prompt information access and collaboration among project personnel in a project. A ubiquitous computing environment will help engineers, project managers, and on-site personnel to more effectively communicate with each other. The first challenge to make such communication effective is to provide project personnel easy access to various engineering information services. In terms of the standard ISO model for communication, we focus our

effort in the application layer rather than the physical access layer. After making the assumption that the communication channel and the network protocol is in place for client devices to gain access to all information services, we have to construct information services in such a way that a wide range of devices with drastically different characteristics can be supported. For example, the output of a CAD design tool must be different on a high-speed graphics workstation from on a Palm handheld device. Information needs to be filtered and presented in different granularities appropriate to the types of client devices. In Section 2, we describe the design and implementation of a mediation-based framework that allows the incorporation of a wide range of computing devices into a distributed engineering service environment.

The second challenge for the ubiquitous environment is to enable effective collaboration among the engineering information services. A given task may require participation from multiple engineering information services. There are two issues involved. First, information is exchanged among various engineering services, for which a common data model is needed. Second, those services need to be coordinated. The execution of a task needs to specify when each service should be invoked, what information is required of each service, and how information is shared among services. We describe a framework and an implementation for seamless interoperation in Section 3. In our demonstration of the integrated services, we use the relational database technology, the eXtended Markup Language (XML) and the Process Specification Language (PSL). We also illustrate in Section 4 an infrastructure where engineering information services can be effectively integrated in the A/E/C industry.

2 Distributed Information Services

Information services need to provide universal accessibility to information clients in ubiquitous computing environments. With software getting more complex and services

becoming more powerful, it becomes essential to define a framework by which software can be constructed to serve clients with dramatically varying computation and communication power. The traditional approach where information services are customized for each client becomes unmanageable when there are a large number of clients, each with different requirements for the services. We introduce a framework for the construction of information services that separates source-specific functionalities from client-specific functionalities. The approach assists the construction and the management of information services that provide universal accessibility.

2.1 Mediation-based Framework

The key challenges in constructing information services are to lower the complexity of software design and to minimize the software maintenance cost. Mediators are introduced to cope with these issues in dynamic collaborative computing environments. Mediators (Wiederhold 1992; Wiederhold and Genesereth 1997) are intelligent middleware that sit between information system sources and clients. They provide integrated information, without the need to integrate the actual data sources. Specifically, mediators perform functions such as accessing and integrating domain-specific data from heterogeneous sources, restructuring the results into object-oriented structures, and extracting appropriate information to be transmitted.

The mediation architecture is conceptually comprised of three layers, as shown in Figure 1. The mediation layer resides between the source access interface and the client access interface, incorporating value-added processing by applying domain-specific knowledge. A major task for an effective mediation service is the reduction of data volume to be shipped to information clients, while maintaining the desired information content. The principal tool for data reduction is abstraction, which increases the granularity. Techniques differ on how the abstraction is obtained and on how the information granularity is controlled.

In traditional mediators, code is written to handle information processing tasks at the time the mediators are built. We call this type of mediator a *static mediator*. Static mediators are used frequently when their behaviors can be established at software construction time. For instance, Wiederhold et al. proposed a static mediator to manage security policies on accessing information sources (Wiederhold et al. 1996). The information clients are each assigned with specific security clearance levels. For each clearance level the security mediator defines the behaviors on how the information content should be processed before the requested information is returned to the client. Once constructed, the mediator will not change its behavior during the course of its service.

As an extension to the static mediators, we introduce active mediators to allow information clients to specify client-defined actions in conducting information processing. Active mediators have the ability to adapt their behaviors to the client requests and the source data streams, providing the ability to dynamically change the granularity of information abstraction (Liu et al. 2000). For instance, an information client can forward a compression routine to the active mediator so that queried information is compressed before returned. For constructing device-independent information services, active mediators are used to dynamically adjust information-processing behavior based on the requests from the information clients.

2.2 Mobile Class

An information-processing module dynamically loaded by the active mediators is called a *mobile class*. Conceptually, a mobile class contains a function that takes (multiple) input data elements, performs some operations on the input, and then outputs a new data element. For instance,

$$y = f(x_1, x_2, x_3)$$

represents a mobile class named *f* that takes three data elements, x_1, x_2, and x_3, as input and produces an output data element *y*.

Java is chosen as the specification language. Mobile classes are implemented in Java language because of Java's support for portability, its flexibility as a high-level language, and its support of dynamic linking/loading, multi-threading and standard libraries. Mobile classes are written in Java and compiled into byte code segments. By incorporating a Java virtual machine, the active mediator can readily execute Java byte codes, thereby enabling the execution of mobile classes on heterogeneous system platforms.

All mobile classes are derived from the *MobileClass* interface, whose definition is shown below:

```
public interface MobileClass {
      public DataElement execute(Vector params);
}
```

The interface contains a single function that represents the functionality of the mobile class. The *execute* function takes a vector of data elements as input and generates a data element as output. The mobile classes overloads the *execute* function to provide specific processing capability. When invoking the mobile class, the active mediator will fill in the content of the input vector with the data obtained from the information sources. The execution of the mobile class is supported by the Java virtual machine incorporated in the active mediator. The *execute* function is invoked, and upon successful execution the mobile class returns the output data element.

There are two types of mobile classes. The call-by-value mobile classes use the parameter-passing scheme where the input parameters contain the values of the input data elements, and the output data element is used to store the processing result of the mobile class. The call-by reference mobile classes, on the other hand, use the parameter-passing scheme where the input

parameters contain the references to the input data elements. The mobile class can directly process the content of the input data elements using the references. The input data elements can be modified, and they represent the processing result of the mobile class. The two types of mobile classes reflect two different programming styles. The active mediator can decide to support either one or both types of mobile classes. The programmers of the mobile classes need to aware of the types of mobile classes that are supported.

Mobile classes can be transmitted from information clients to active mediators for execution because of their support for heterogeneous platforms. Information clients therefore can utilize mobile classes to conduct complex processing on the remote site of active mediators. For instance, mobile classes can be used for data compression, data expansion, aggregation, relational operations, etc. Figure 2 shows a sample mobile class *PalmTrimmer* that conducts information extraction for clients using handheld devices. *PalmTrimmer* is designed for an active mediator that integrates project information from multiple information sources. The execution of *PalmTrimmer* results in trimming the project activity information, hence allowing the condensed information to be displayed by the handheld devices. *PalmTrimmer* is a derived class of *MobileClass*, whose *execute* function is overloaded to provide the specific extraction functionality. *PalmTrimmer* uses a call-by-reference parameter passing scheme, where the first element in the input parameter vector for the *execute* function is the reference to the data fetched from the information source. The active mediation runtime environment fills in this element handle when the mediator loads the mobile class. *PalmTrimmer* operates directly on the data element and removes everything other than the activity identifiers and the activity descriptions from the data element. The *execute* function does not need to return any data since the processed result is accessed by reference.

Major benefits ensue from the ability to migrate the mobile classes from information clients to information sources. First of all, mobile classes provide flexibility to the information service providers, who no longer need to specify all the information processing functionalities when building the mediator. Compared to the static mediation where any changes in the information processing functionalities involves code modifications to the mediator, active mediation with mobile classes alleviates the maintainer of the mediator much of that burden. Client specific functionalities are specified using mobile classes, whereas functionalities that are shared by all information clients, such as source information query, are specified within the mediator. This approach makes the development and the maintenance of the mediators more manageable. Secondly, mobile classes can provide performance improvement to the overall information system. By selecting the best locations for the executions of the mobile classes, from the sites of either the information clients or the information sources, the system can reduce the overall data volume transmitted among various components of the system. Rather than sending data to be executed at where the code is located, mobile classes enable code to be sent to where data is located. When the benefit gained by the reduction of data communication traffic outweighs the cost of migrating code, the system performance improves.

There are many methods to create mobile classes. We have developed templates based on which mobile classes are specified to perform simple schema-based information filtering. Template-based information processing techniques can be found in many other technologies, such as XSLT technology, which allows user to specify style-sheets for transforming an XML document into another XML document (Clark 1999). The template-based approach benefits from its relative simplicity. However, templates only offer very limited functionalities, which focus mainly on string based searching and text matching. Mobile classes provide richer

functionalities than templates. For instance, logical and arithmetic operations become handy in the Java programming language, giving the mobile classes the ability to perform operations such as complex logical comparisons and aggregations that are important for information abstraction. Our approach allows programmers to use templates for developing mobile classes skeletons, which can be further extended with more complex functionalities.

2.3 Active Objects

Software applications collaborate by exchanging information. A project-scheduling program may need to obtain schedule information from a modeling tool and send the analysis results to an information retrieval program. The lack of a reliable, simple and universally deployed data exchange model has long impaired effective interoperations among heterogeneous software applications. To achieve data interoperability, applications typically need to map their data models and formats to other applications, requiring what is often called 'legacy wrapping' (Hammer et al. 1995). There are several problems associated with this approach. First, every connection between two applications will most likely require custom programming. For each pair of applications, a custom wrapper needs to be built. If many applications are involved, a lot of programming effort will be needed. Furthermore, maintenance of the custom wrappers is very expensive. Any data model and format changes in an application will affect all wrappers that have one end connecting to the application. Also, data corruption and parameter mismatch can cause unpredictable results, and debugging and error handling becomes difficult since many wrappers need to be looked at simultaneously. Because of its fragility, legacy wrapping incurs high maintenance cost.

The notion of objects developed in the object-oriented programming methodology can be effectively utilized for communicating information between various applications, as

demonstrated previously for mediators (Papakonstantinou et al. 1996) and for the building industry (Snyder and Flemming 1999). When the underlying resources are modeled as objects, the connections among the resources can be encapsulated. Objects can be represented in many data formats. Although XML is not strictly object-oriented, we choose XML as our information representation format based on XML's nature of extensibility, structure, and validation as a language . As a simple textual language, XML is quickly gaining popularity for data representation and exchange on the Web. XML is a meta-markup language that consists of a set of rules for creating semantic tags used to describe data. An XML element is made up of a start tag, an end tag, and content in between. The start and end tags describe the content within the tags, which is considered the value of the element. In addition to tags and values, attributes are provided to annotate elements. Thus, XML files contain both data and structural information. In essence, XML provides the mechanism to describe a hierarchy of elements that forms the object.

An active object is a special type of XML object. In active objects, two types of elements are defined: data elements and active elements. A data element is a regular XML element that describes the structure and data contents of an object. An active element, on the other hand, no longer describes the content of an object but rather contains a mobile class that can be applied to conduct information processing. The mobile class is presented in the form of serialized byte codes, encoded in legal string characters for XML documents. The mobile class will be decoded into normal Java byte codes before being invoked by an active mediator.

Figure 3 shows a sample active object that contains an active element. The active object specifies a query request for activity information from an information service. The active element named *PalmTrimmer* contains a mobile class, which is identified by setting the *active-node* attribute to “yes”. The Java source code of the mobile class is shown in Figure 2, and the

compiled byte code is enclosed in the active element. When active mediator processes the request, the active object will be separated into two components. The string "*ACTIVITY*" specified in the active object is used to query the information sources, which return the result in the form of an XML object. The mobile class is then used to extract useful information from the XML object and return the filtered information to the client.

Active object is a key component of active mediation technology. It enables information clients to define dynamic code segments that can be used by information services to conduct client-specific information processing. The responsibility to provide client-agnostic information content remains with source information service, while the responsibility to define client-specific information abstraction and reduction are shifted to individual information clients.

2.4 Active Mediator Architecture

The active mediator is an information-processing engine that resides between source information services and information clients. The key feature of an active mediator is its ability to handle active objects, using which the information clients can expand the functionalities of an active mediator. Figure 4 illustrates the architecture of an active mediator, which conceptually consists of four functional units and two code segment repositories:

1. The **Mobile Class Handler** is responsible for identifying mobile classes and decoding mobile class byte codes.
2. The **Mobile Class Cache** is a temporary storage for the Java byte codes of mobile classes. The cache is used to avoid duplicate loading of the mobile classes. The byte codes of a mobile class are first looked up from the mobile class cache. When a cache miss occurs, the mobile class fetcher is used to load the byte codes.

3. The **Mobile Class API Library** stores utility classes that make the construction of mobile classes more convenient. For instance, the Java Development Kit library (Arnold et al. 2000) is provided as part of the mobile class API library.
4. The **Mobile Class Runtime** is the module where mobile classes are executed. The runtime invokes appropriate mobile classes to conduct dynamic information processing.
5. The **Exception Handling** module provides a comprehensive set of policies to handle abnormalities in loading or processing mobile classes. Our current implementation prohibits any results from getting through the mediator in the case of an exception. In addition, the conditions are logged for future maintenance.
6. The **Data Mediator** provides information integration for the underlying information sources. It provides the functionalities that a static mediator would provide, incorporating information-processing logic specific to the application domain.

The Mobile Class Handler is the functional unit that processes all incoming client requests, dividing active object requests into queries and mobile classes. The queries are forwarded to the Data Mediator and the mobile classes are decoded and stored into the Mobile Class Cache. Queries received by the Data Mediator are reformulated into source queries based on the domain knowledge acquired by the mediator, and the source queries are forwarded to the source information services. The data returned from the services are represented as XML objects, which are further integrated according to the domain specific logic incorporated in the Data Mediator. The integrated objects are then forwarded onto the Mobile Class Runtime. The runtime environment loads relevant mobile classes from the Mobile Class Cache and the Mobile Class API Library. The mobile classes are invoked with their execute function to process the objects. Finally, the results are returned to the information client.

The active mediation system can be deployed without changes to either the information client or the source information service, enabling a smooth transition from a legacy information service infrastructure to one using active mediation framework. The Data Mediator functions as the data integrator for the source information services. A regular query without enclosed mobile classes will simply flow through the components of the active mediator without invocation of any mobile classes, in which case the active mediator has the same function as a static mediator.

Conceptually, the active mediation layer resides between source information services and information clients. In practice, the active mediation layer may either be separated from or be combined with other layers. Figure 5 illustrates two alternative approaches when deploying active mediation framework. The first approach, as shown in Figure 5(a), makes active mediation a separate layer from the source information service. To warrant implementation of the mediation service as a distinct module, there must be sufficiently much added value to overcome the cost of adding a layer and its interfaces in the information processing flow. On the other hand, the benefits and costs to be considered are only partially related to performance. Having identifiable and maintainable service modules provides significant long-term management benefits.

When the source service interfaces are not well defined, alternatively, we could build active mediation into the source information service, as illustrated in Figure 5(b). Building a separate layer of active mediation requires source services to expose their functionalities through external interfaces, including those that are not required by information clients but needed by the active mediators. The cost of modifying the existing source services may prove to be very high and outweigh the benefit of modularity and manageability brought by a separate active mediation layer. Building the active mediation inside the source services enables active mediator direct

access to the functionalities offered by the services, both internal and external to the services, and may thus be the best engineering choice.

2.5 Device-Independent Information Services

Information services usually cannot adjust their behaviors to clients with different characteristics and requirements. Most services are designed with a specific type of clients in mind. Retrofitting existing services to be device-aware is a very expensive exercise; it goes against the good software design principle of separating client specification and server functionality. Moreover, it is infeasible to cover all existing devices and to foresee all future client device types. The key in constructing device-independent information services thus lies in separating the information content that a service provides from the presentation of the content.

Active mediation is a natural solution that gives users of information services the ability to specify how information should be processed. As shown in Figure 6, many clients with different characteristics may be requesting information from the same set of information services. The clients require different granularity and presentation for the same piece of information. This is achieved by having clients provide information processing routines in the form of mobile classes. An intermediate active mediation layer is inserted between the information clients and the source services. The active mediator invokes the clients' mobile classes on the source information and thereby conducts client-specific information processing, including data reductions, mappings and adaptations. Responsibility for the output content filtering is thus shifted to those who actually know what information they want to see.

The source information services remain device-independent. The services provide answers to source queries that contain no explicit identifying information about the types of the clients. The information content is passed to the active mediator for further processing. As an example,

information content may contain data in the form of XML objects and presentation styles in the form of XML style sheet. The active mediator invokes appropriate mobile classes to process the information content obtained from the source services. The processed content is then returned to the clients.

We partition the process of constructing device-independent information services into two phases using the active mediation framework. The first phase focuses on constructing source services to provide modular and object-oriented information content. The second phase focuses on developing information processing routines for each client device type. As client devices have drastically different characteristics, they require different levels of abstraction for information content. For instance, a handheld Palm device requires high abstraction levels so that content can be transferred through a thin communication channel and presented on relatively small screen space. In contrast, a high-speed workstation digests and presents more information, therefore requires less abstraction. Certainly, the design process is iterative. Feedback loops are necessary to adjust source information services so that information abstraction can be done more effectively and efficiently.

New client device types can be added into the existing computing environment by developing new information processing routines. No modifications are necessary in either the source information service, or the active mediator, or the other client devices. With active mediation, we can develop device-independent information services that provide universal accessibility to current and future information clients.

3 Integration of Information Services

When multiple information services are used collaboratively, effective interaction among information services is an important aspect of ubiquitous computing. Information services need

to be integrated so that they can exchange information despite differences in how their data is represented. In addition, when multiple information services are involved with a given task, a coordination mechanism needs to be in place so that services can be invoked in appropriate order and their results directed to receiving services for further processing.

3.1 Information Modeling

Information modeling plays an important role in distributed service integration. Information in different applications usually has different representations. Even for the same type of application, the internal representations of the information are also different. To cope with different representations among applications, we need an ontology standard to model information. Ontology refers to a collection of terms and their relationships, which enable consistent communication in a specific domain.

There have been many efforts to develop product data standards for data exchange in the Architecture/Engineering/Construction industry, such as STEP (Mason 1991), IFC (IAI 1997), ifcXML (Liebich 2001), aecXML, etc. Most current ontology standards, however, focus mainly on product data; they do not provide extensive information about process and task specifications that are important data attributes for project management applications.

3.1.1 Process Specification Language

PSL (Process Specification Language) was initiated by NIST (National Institute of Standards and Technology) and is emerging as an international standard for process representation (Schlenoff et al. 2000). The goal is to create a language for the exchange of process information among different applications. Two basic reasons motivate the development of PSL. First, there are few existing standards for exchanging process information. Second, current ontology

standards lack a formal logic to define relationships and constraints. PSL is based on first order logic and situation calculus; they make it an ideal candidate standard for representing process information and managing project and workflow.

PSL is based on KIF (Knowledge Interchange Format), which is designed for knowledge interchange among disparate computer systems (Genesereth and Fikes 1992). KIF has declarative semantics, and is logically comprehensive. Figure 7 shows the overall organization of PSL, which includes the PSL core, the PSL outer core and PSL Extensions (Schlenoff et al. 2000):

- The PSL core is a set of axioms based on KIF. The PSL core includes four basics classes: Object, Activity, Activity_Occurrence and Timepoint. Relations are defined among the classes, for example:

  (occurrence-of activity-occurrence activity)

  (before timepoint timepoint)

- PSL outer core consists of a small set of generic extensions, including Subactivity Extension, Activity-Occurrence Extension and States Extension. For example, relations may be defined using the PSL outer core extensions as:

  (subactivity-occurrence activity-occurrence activity-occurrence)

  (subactivity activity activity)

- PSL extensions include ontology modules such as generic activities, ordering relations and schedules. Each module is motivated by a set of applications and covers concepts in a specific application domain. Below are some example relations in the PSL extensions:

  (before-start activity-occurrence activity-occurrence activity-occurrence)

(before-start-delay activity-occurrence activity-occurrence activity-occurrence duration)

We have extended the PSL core by including extensions that model the essential information related to project management applications (Cheng and Law 2002). Due to its logic framework, PSL can potentially be used to check consistency of project information by using a reasoning tool (Cheng et al. 2003).

3.1.2 Implementation of PSL Wrappers

Once the PSL ontology for a specific application domain is defined, software wrappers, which act as a bridge between common (PSL) representation and proprietary representations (for each application), need to be built. PSL wrappers are used to retrieve project information from the applications, and are also used to update project information in these applications. The basic process of using PSL for project information exchange can be illustrated in Figure 8 and consists of three major steps -- ontology mapping, communicating with applications, outputting or parsing PSL files. It is not unusual that the same term is often associated with different meanings in different applications. To exchange project information, first we need to map the concepts in different applications into PSL ontology, so that they are PSL compliant.

Different wrappers are developed to transfer and retrieve information to and from different applications. The application software considered in our current prototype infrastructure includes Primavera P3™, MS Project™, and 4D Viewer (McKinney and Fisher 1998). The applications can exchange information using PSL as the ontology standard. To enhance the accessibility of the project information from these applications, we also build a translator between PSL and database. We have designed a database schema according to the PSL ontology

and developed a translator in Java to convert information from database to PSL file and vice versa.

3.2 Coordination of Distributed Information Services

Information services collaborate to accomplish a task. A couple of issues need to be resolved for the functionalities from individual services to be composed together. First, information services need to be provided in such a way that their functionalities can be accessed and composed. Second, an environment is needed to support the composition and execution of the collaborating services.

3.2.1 Composition of Megaservices

The vision of composing functionalities from multiple software services is echoed by the megaprogramming framework, where a composed service is called megaservice (Boehm and Scherlis 1992; Wiederhold et al. 1992). A megaservice specifies the actions and relationships among the involved information services. Though distributed and heterogeneous, information services can be utilized as if they were locally available to the megaservice.

Megaservices automate the executions of the involved information services that collaborate by sharing information. Figure 9 shows a sample megaservice that involves three engineering information services. The megaservice retrieves a project model using the *ModelRetriever* service, then conducts scheduling on the model using the *Scheduler* service, and finally notifies the related parties about the change via the *ChangeManager* service. The shared information passed around the services is identified by the common parameter names. For instance, the project model fetched by the *ModelRetriever* service is passed to the *Scheduler* service as an input parameter.

Software applications are often managed under distributed and heterogeneous administrative environments. They run on different hardware and software platforms, and their interfaces use different data formats and network protocols. To facilitate service composition, software applications are wrapped into information services that employ a homogeneous model to facilitate communications and collaboration.

The FICAS (Flow-based Infrastructure for Composing Autonomous Services) metamodel (Liu et al. 2002a) is chosen as our framework to build information services. The key characteristic of the FICAS metamodel is the explicit separation of control-flows from data-flows, which respectively represent groups of related control messages and data messages used to exchange information among information services. Control messages and data messages are distinguished by their use at the recipients of the messages. Control messages are mostly short messages that trigger state changes at the receiving services. Data messages are mostly large data packets that are given to the receiving services for processing. For data-flow, the information service primarily concerns about performing services on the input data and generating output data. For control-flow, the information service primarily concerns about the state management of the service, e.g. the completion of a task, the termination of a service, etc.

3.2.2 Service Composition Infrastructure

Service composition infrastructure is responsible for composing and executing megaservices. FICAS is a service composition infrastructure that supports distributed data-flows (Liu et al. 2002a). As shown in Figure 10, FICAS consists of buildtime and runtime components. The buildtime components are responsible for composing megaservices and compiling megaservice specifications into control sequences that serve as inputs to the runtime environment. The runtime components are responsible for the executions of the control sequences.

Composition of autonomous services starts with the megaservice specification. We have defined the CLAS language to provide the application programmers the necessary abstractions to describe the behaviors of their megaservices (Liu et al. 2002b). The CLAS program is translated by the buildtime component into a control sequence that can be executed by the runtime environment. The control sequence is language and platform independent, providing a bridge between megaservice specification and megaservice execution.

The FICAS runtime environment is responsible for executing the control sequences. The megaservice controller is the entity that carries out the execution of a megaservice. The controller first converts an input control sequence into an execution plan, and then follows the plan to coordinate control-flows among the respective services. The controller serves as the centralized coordinator for all the control messages incurred by the megaservice. Since the megaservice execution is carried out with parallel invocations of autonomous services, the controller is also responsible for synchronizing control-flows and conducting performance optimization. The service directory is created to index the information service parameters. It keeps track of available services within the infrastructure. The directory is viewed globally as a centralized entity, while it may be implemented as a distributed structure.

FICAS is chosen as the service composition infrastructure to support our ubiquitous computing environment because it can address the key issues in composing megaservices:

(1) Ease of composition – The compositional language CLAS provides an effective and convenient mechanism to application programmers for specifying compositions of functionalities provided by individual information services.

(2) Scalability – The metamodel used by the information services allows services to be independently constructed and plugged into the service composition infrastructure, thus facilitating the integration and management of large number of services.

(3) Performance – The runtime environment employs the distributed data-flow model, hence avoiding the communication bottleneck at the megaservice controller. The runtime can deliver high performance for the execution of megaservices composed of engineering information services.

4 Ubiquitous Computing Environment

In this section, we illustrate our implementation of an engineering information service infrastructure for the A/E/C industry. The technologies described in prior sections are utilized to deliver ubiquitous computing. An engineering scenario is then described to demonstrate how the service infrastructure allows prompt information access and facilitates collaborations among project personnel.

4.1 Integrated Service Infrastructure

For a typical construction project scenario, different construction applications can reside at different locations. Some applications may reside on the site offices, and others may reside in the company headquarter. Project information is not shared and accessible among all project participants at all time. It is difficult and time-consuming for project managers on the construction sites to get the latest project information from the company headquarter and other places. If there are some changes on the construction site, it is also hard for project managers to evaluate the impacts of the changes on the whole project immediately, since the project mangers may not have access to project management application and 3D/4D application on the

construction site. Project managers cannot reschedule the project on the construction site right away using the latest information. To improve the proficiency, a ubiquitous environment is highly desired, so that project participants can access and manage the latest project information from various engineering services, using different software applications and at different locations.

An infrastructure shown in Figure 11 has been developed to illustrate how the ubiquitous computing environment is developed for distributed project management services. There are five types of clients and applications involved in the environment. The Palm PDA devices are used to access project information via wireless modems. The web browsers provide project information to users who usually have access to high-speed Internet connections and more powerful computing devices. Three engineering software applications are included to manage the design and scheduling aspects of the project. 4D Viewer is an effective tool for analyzing and visualizing 3D architectural designs and their relationships to project schedules (Koo and Fischer 2000). Primavera P3 is a specialized tool that focuses on the scheduling aspect of the project. And, Microsoft Project is another tool used for managing project schedules.

An Oracle 8i relational database serves as the backbone information store for this distributed service infrastructure. The active mediator acts as an intelligent bridge that connects various devices with the database. It captures the client requests in the form of active objects from devices such as Palm and desktop browsers. Source queries are constructed and sent to the Oracle 8i database. The active mediator retrieves the information from the Oracle 8i database and conduct client-specific information processing by invoking the mobile classes incorporated in the client requests. The processed information with desired abstraction and suitable format is returned to the clients for displaying.

The engineering software applications all have different proprietary data models for describing project schedule information. We use PSL as a common data model, through which the different applications can communicate with each other. PSL data converter also acts as a bridge to map between the proprietary data models and the relational data model, enabling Oracle 8i database to serve as the backbone information store.

Functionalities from various software applications can be further composed when the applications are wrapped as information services and their functionalities exposed via well-defined interfaces. For instance, Primavera P3 is wrapped into a *P3 Scheduling Service*, which provides functionalities such as project rescheduling. PSL data converter is wrapped into the *PSL Model Service*, which provides functionalities such as extracting PSL project models from the Oracle 8i relational database and storing PSL models into the database. As shown in Figure 12, the services are composed into the rescheduling megaservice. The *PSL Model Service* is first used to fetch the project model from the Oracle 8i relational database. The model is then forwarded onto the *P3 Scheduling Service* for rescheduling. The rescheduled project model is then stored back to the relational database by invoking the *PSL Model Service* again. Finally, a change notification service is invoked to inform all relevant parties of the change to the project.

4.2 An Engineering Scenario

We now look at an example scenario and demonstrate how the ubiquitous computing environment may help facilitate personnel from different functional groups conduct collaborations. We use the project model of the Disney Concert Hall as the test case example. Figure 13 shows a snapshot of the construction progress using the 4D Viewer. Figure 14 is the view of the scheduling information using Primavera P3. Using PSL as the intermediate data model, the information is shared between the relational data model and the proprietary Primavera

data model. The scheduling information can also be reviewed using a handheld Palm device, for example by on-site personnel, as shown in Figure 15. The information is first converted into XML model, and then the active mediator filters the information and adapts the content for the handheld device that is reviewing the information.

Suppose, as a hypothetical example, that the duration for the activity, 18T1-33201, for erecting a roof element is to be changed from 1 day to 40 days (see Figure 15). The change can be made remotely using the Palm device by on-site personnel. The update will be stored into the relational database and trigger the rescheduling megaservice. The revised schedule can be viewed using MS Project as shown in Figure 16 and the project model can be displayed and viewed using the 4D Viewer as shown in Figure 17. The project status can also be viewed using a simple web browser as shown in Figure 18. The web browser adopts the same information path as in the case of the Palm device. An XML model constructor and an active mediation are used to generate appropriate information content for different information clients. Comparing to the Palm device, the web browser can display much more detailed scheduling information, for example, with the altered activity and the affected activities highlighted using different colors – the information that project managers may find helpful to diagnose the impact of the updated schedule.

5 Summary

We have identified two main issues in enabling ubiquitous computing environment for engineering services: accessibility and collaboration. We address the problems by acquiring methodologies to develop services and build service infrastructures that provide universal accessibility and promote interactivity among existing engineering services.

Active mediation is introduced as a value-added service layer that resides between source information service and information client. It provides the information clients the ability to specify information processing routines according to the characteristics of the client devices, hence making source information service device-independent. We describe the architecture of the active mediator and discuss the alternative methods for implementing active mediation services.

We have shown how PSL, XML and relational models can all be used together to effectively model data used in various engineering service tools. By building a PSL wrapper for each application, we can exchange project information successfully using PSL as a standard. By building a translator between PSL and Oracle database, we greatly improve the accessibility of the project information in various engineering services. As PSL is based on first order logic and situation calculus, we expect that PSL play a more important role in workflow management than an interchange standard.

In addition to effective data integration among different software tools, ubiquitous computing requires truly integrated engineering services. We define a metamodel for information services, based on which services can be accessed and composed in a homogeneous manner. Information services can then be composed into megaservices that provide integrated functionalities. A distributed data-flow service composition infrastructure is introduced to provide support for the composition and the execution of the composed services.

6 Acknowledgement

This work is partially sponsored by the Center for Integrated Facility Engineering at Stanford University, a Stanford Graduate Fellowship, the Air Force (Grant F49620-97-1-0339, Grant F30602-00-2-0594), and the Product Engineering Program at NIST. The Product Engineering

Program gets its current support from the NIST's SIMA (Systems Integration for manufacturing Applications) program. The 4D Viewer and the 4D model of the Mortenson Ceiling Project are provided by Professor Martin Fischer and his research group at Stanford University. No approval or endorsement of any commercial product by the National Institute of Standards and Technology or by Stanford University is intended or implied.

7 References

Arnold, K., Gosling, J., and Holmes, D. (2000). *The Java Programming Language*, Addison-Wesley, Boston, MA.

Boehm, B., and Scherlis, B. (1992). "Megaprogramming." *DARPA Software Technology Conference*, Los Angeles, 68-82.

Cheng, J., and Law, K. H. (2002). "Using Process Specification Language for Project Information Exchange." *3rd International Conference on Concurrent Engineering in Construction*, Berkeley, CA, 63-74.

Cheng, J., Law, K. H., Gruninger, M., and Sriram, R. D. (2003). "Process Specification Language For Project Information Exchange." *International Journal of Information Technology in Architecture, Engineering and Construction*, in press.

Clark, J. (1999). "XSL Transformations (XSLT) Version 1.0." W3C Recommendation, World Wide Web Consortium.

Genesereth, M. R., and Fikes, R. (1992). "Knowledge Interchange Format Reference Manual - Version 3." *CSD-Logic-92-1*, Stanford University.

Hammer, J., Garcia-Molina, H., Labio, W., Widom, J., and Zhuge, Y. (1995). "The Stanford Data Warehousing Project." *Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing*, 18(2), 41-48.

IAI. (1997). "Industry Foundation Classes." *Release 2*, International Alliance for Interoperability, Washington, DC.

Koo, B., and Fischer, M. (2000). "Feasibility Study of 4D CAD in Commercial Construction." *Journal of Construction Engineering and Management, ASCE*, 126(4), 251-260.

Liebich, T. (2001). "XML Schema Language Binding of EXPRESS for ifcXML." *MSG-01-001*, International Alliance for Interoperability.

Liu, D., Law, K. H., and Wiederhold, G. (2000). "CHAOS: An Active Security Mediation System." *International Conference on Advanced Information Systems Engineering*, 232-246.

Liu, D., Law, K. H., and Wiederhold, G. (2002a). "Data-flow Distribution in FICAS Service Composition Infrastructure." *15th International Conference on Parallel and Distributed Computing Systems*, Louisville, KY.

Liu, D., Law, K. H., and Wiederhold, G. (2002b). "FICAS: A Distributed Data-Flow Service Composition Infrastructure." <http://mediator.stanford.edu/papers/FICAS.pdf>, (Dec. 15, 2002).

Mason, H. (1991). "Industrial Automation Systems -- Product Data Representation and Exchange -- Part 1: Overview and Fundamental Principles, Version 9." *TC184/SC4/WG PMAG Document N50*, ISO.

McKinney, K., and Fisher, M. (1998). "Generating, Evaluating and Visualizing Construction Schedules with CAD Tools." *Automation in Construction*, 7(6), 433-447.

Papakonstantinou, Y., Abiteboul, S., and Garcia-Molina, H. (1996). "Object Fusion in Mediator Systems." *VLDB Conference*.

Schlenoff, C., Gruninger, M., Tissot, F., Valois, J., Lubell, J., and Lee, J. (2000). "The Process Specification Language (PSL): Overview and Version 1.0 Specification." *6459*, National Institute of Standards and Technology, Gaithersburg, MD.

Snyder, J., and Flemming, U. (1999). "Information Sharing in Building Design." *8th International Conference on Computer Aided Architectural Design Futures*, 165-183.

W3C. (1996). "Extensible Markup Language (XML)." <http://www.w3.org/xml>.

Wiederhold, G. (1992). "Mediators in the Architecture of Future Information Systems." *IEEE Computer*, 38-49.

Wiederhold, G., Bilello, M., Sarathy, V., and Qian, X. (1996). "A Security Mediator for Healthcare Information." *1996 AMIA Conference*, 120-124.

Wiederhold, G., and Genesereth, M. (1997). "The Conceptual Basis for Mediation Services." *IEEE Expert, Intelligent Systems and Their Applications*, 12(5), 38-47.

Wiederhold, G., Wegner, P., and Ceri, S. (1992). "Towards Megaprogramming." *Comm. ACM*, 35(11), 89-99.

List of Figures

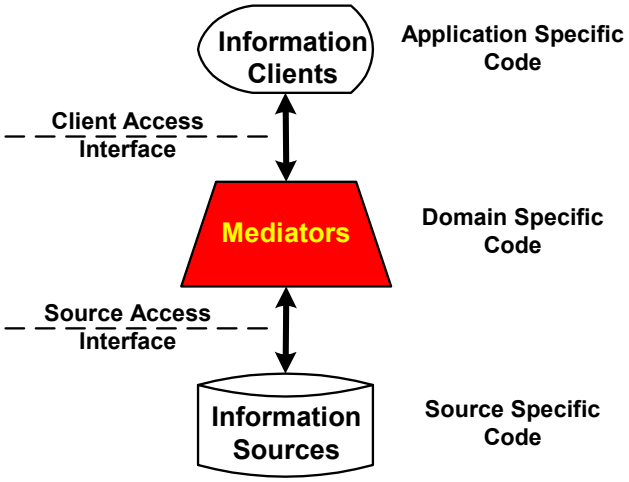

Figure 1: Mediation Architecture

```
public class PalmTrimmer implements MobileClass
{
  /* params contains root element as its first element */
  public DataElement execute(Vector params) {
        Element root = (Element) params.firstElement();

        Vector tags = new Vector();

        tags.addElement(new String(“ACTIVITYID"));
        tags.addElement(new String(“DESCRIPTION"));

        keepOnlyNodes(root, tags);

        return null;
     }
}
```

Figure 2: Mobile Class PalmTrimmer

```
<REQUEST>
  <QUERY>
    ACTIVITY
  </QUERY>
  <PalmTrimmer active-element="yes">
    ...
    encoded byte code for PalmTrimmer mobile class ...
    ...
  </PalmTrimmer>
</REQUEST>
```

Figure 3: A Sample Request Active Object and Active Node Source Code

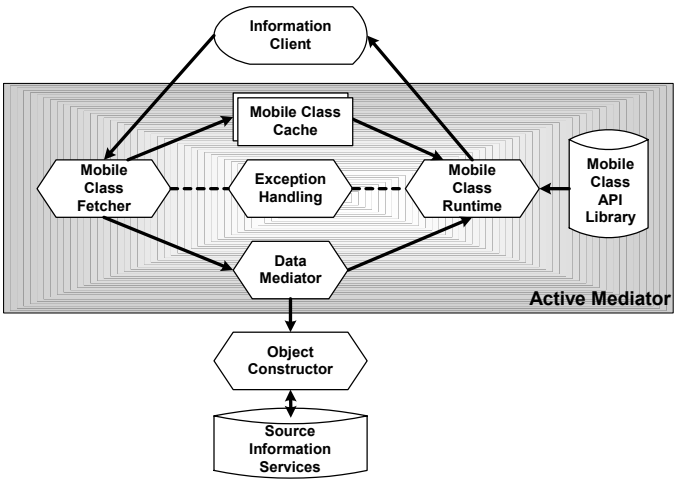

Figure 4: Active Mediation Architecture

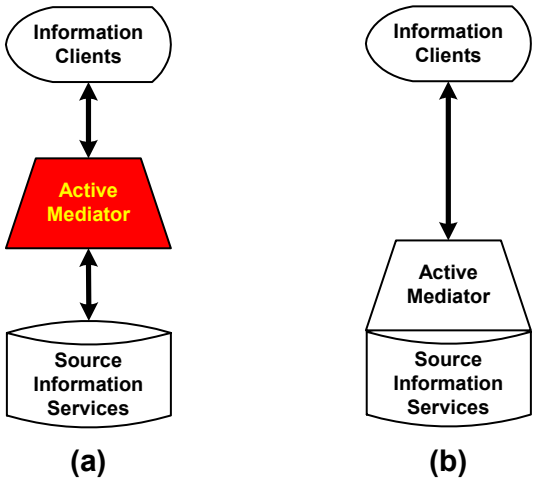

Figure 5: Incorporation of Active Mediation with Information Services

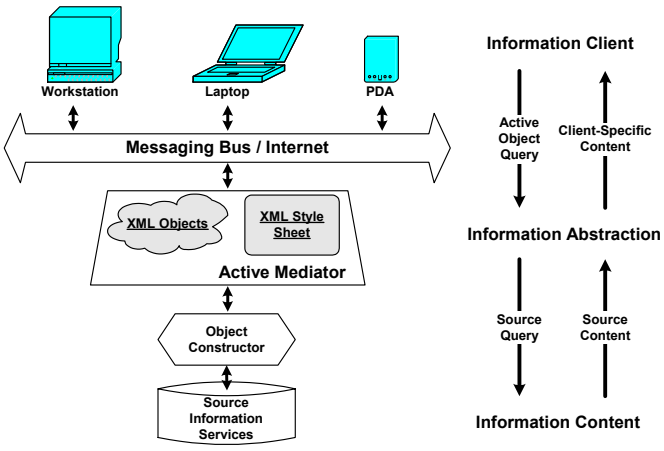

Figure 6: Active Mediation in Information Service Construction

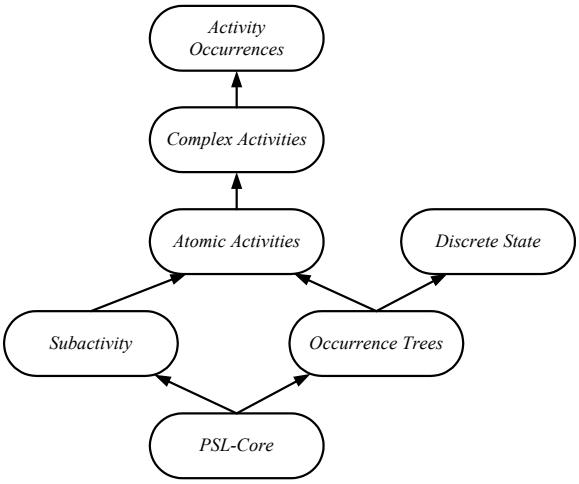

Figure 7: PSL Ontology

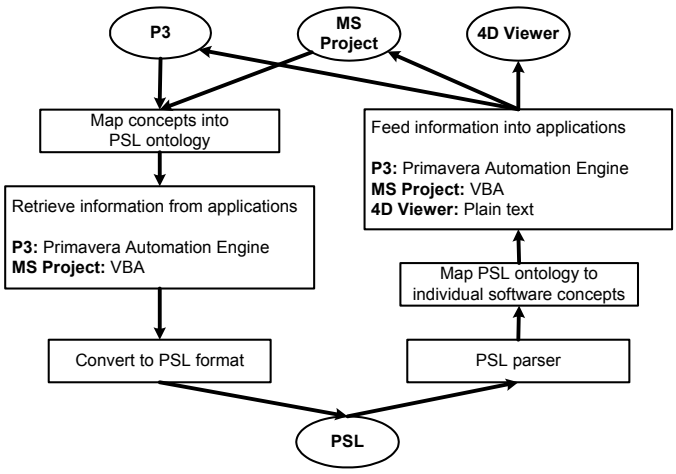

Figure 8: PSL Wrappers

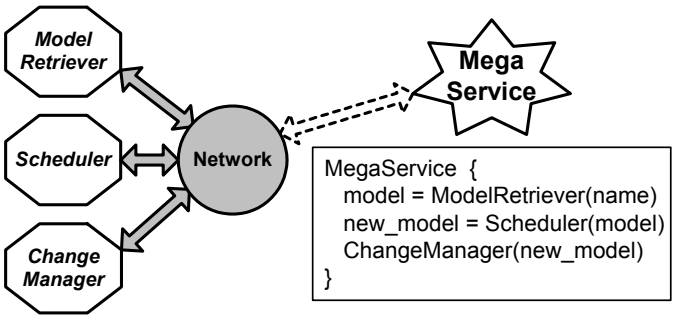

Figure 9: A Megaservice for Engineering Services

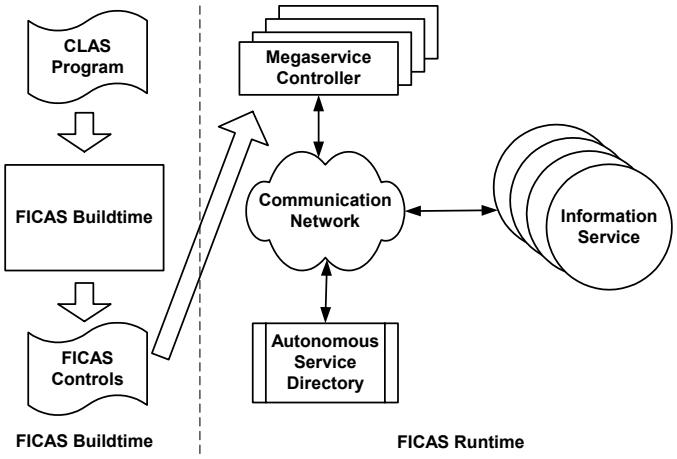

Figure 10: FICAS Architecture

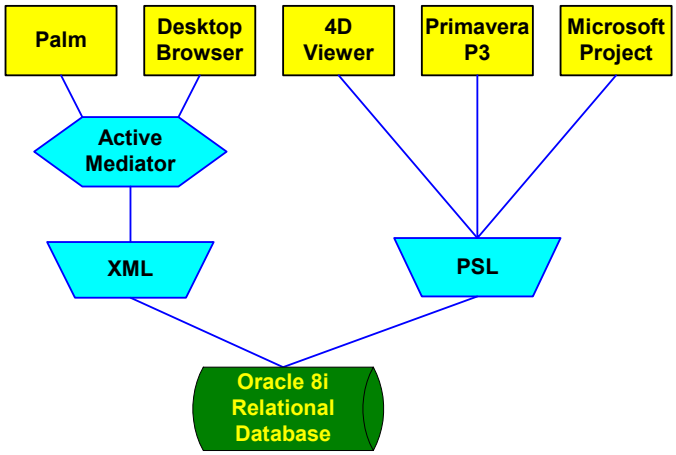

Figure 11: Software Integration for the Ubiquitous Computing Environment

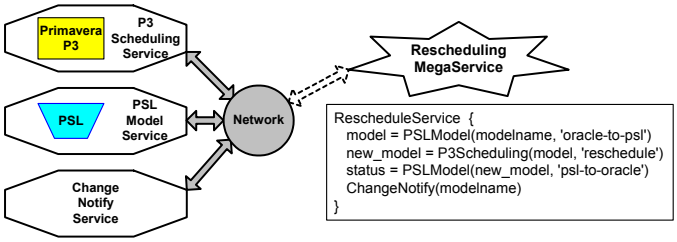

Figure 12: Rescheduling Megaservice

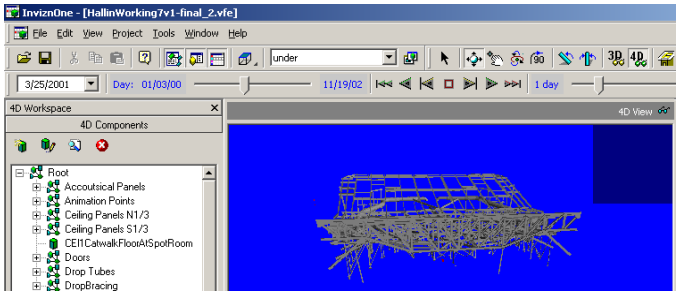

Figure 13: Reviewing Sample Project on 4D Viewer

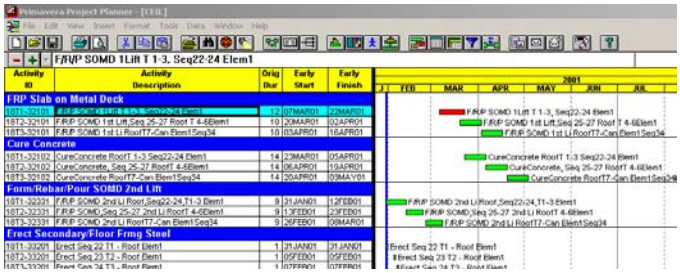

Figure 14: Reviewing Sample Project on Primavera

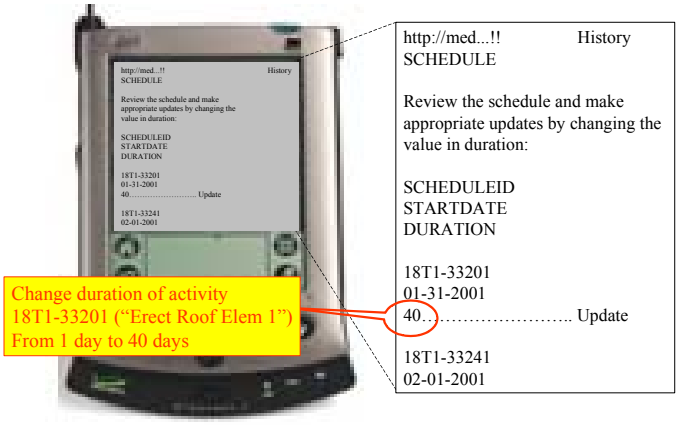

Figure 15: Revising Project Schedule Via a Palm Device

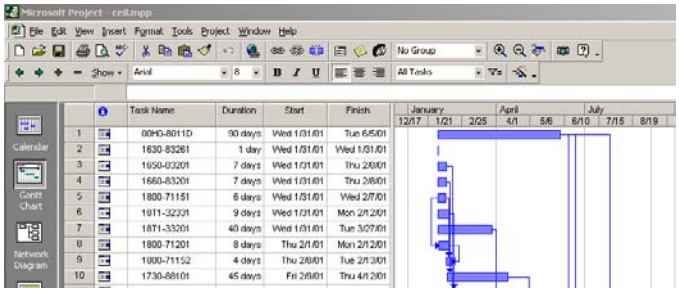

Figure 16: Regenerated Gantt Chart in Microsoft Project

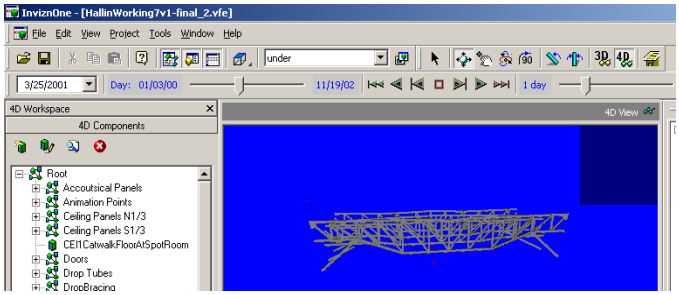

Figure 17: Reviewing Updated Project on 4D Viewer

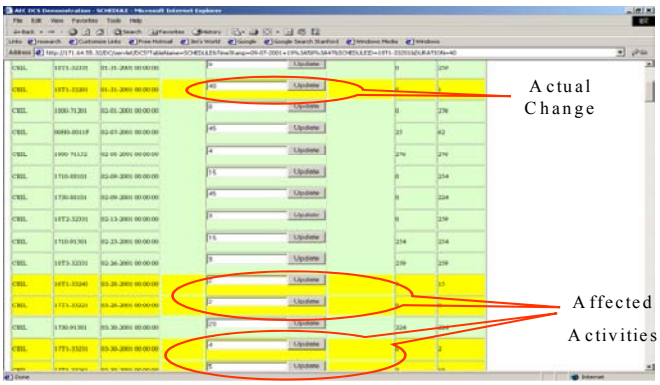

Figure 18: Reviewing Updated Schedule on Web Browser